Introduction to Applied Scientific Computing using MATLAB

Mohsen Jenadeleh

In this lecture, slides from Mathworks, MIT, Waterloo and Rutgers Universities are used

Our Guiding Principles

"The purpose of computing is insight, not numbers"

Richard Hamming

"I hear and I forget,
I see and I remember,
I do and I understand."

Confucious

Main Features of MATLAB

- Easy and efficient programming in a high-level language, with an interactive interface for rapid development.
- Vectorized computations for efficient programming, and automatic memory allocation.
- Built-in support for state-of-the-art numerical computing methods.
- Has variety of modern data structures and data types, including complex numbers.
- High-quality graphics and visualization.

- Symbolic math toolbox for algebraic and calculus operations, and solutions of differential equations.
- Portable program files across platforms.
- Large number of add-on toolboxes for applications and simulations.
- Huge database of user-contributed files & toolboxes, including a large number of available tutorials & demos.
- Allows extensions based on other languages, such as C/C++, supports Java and object-oriented programming.

MATLAB Toolbox Application Areas

- Parallel Computing (2)
- Math, Statistics, and Optimization (8)
- Control System Design and Analysis (6)
- Signal Processing and Communications (7)
- Image Processing and Computer Vision (4)
- Test and Measurement, Data Acquisition (5)
- Computational Finance, Datafeeds (7)
- Computational Biology (2)
- Code Generation and Application Deployment (11)
- Database Connectivity (2)

(54 toolboxes + 35 simulink products)

Web Resources

- Getting Started with MATLAB (HTML)
- Getting Started with MATLAB (PDF)
- MATLAB Examples
- MATLAB Online Tutorials and Videos
- MATLAB Interactive Tutorials
- MATLAB Toolbox Reference Manuals
- MATLAB Interactive CD
- Newsletters

- MATLAB User Community
- Other MATLAB Online Resources
- comp.soft-sys.matlab newsgroup

- Octave – a free look-alike version of MATLAB
- FreeMat – another free look-alike version

- NIST – Digital Library of Mathematical Functions
- NIST – Physical Constants

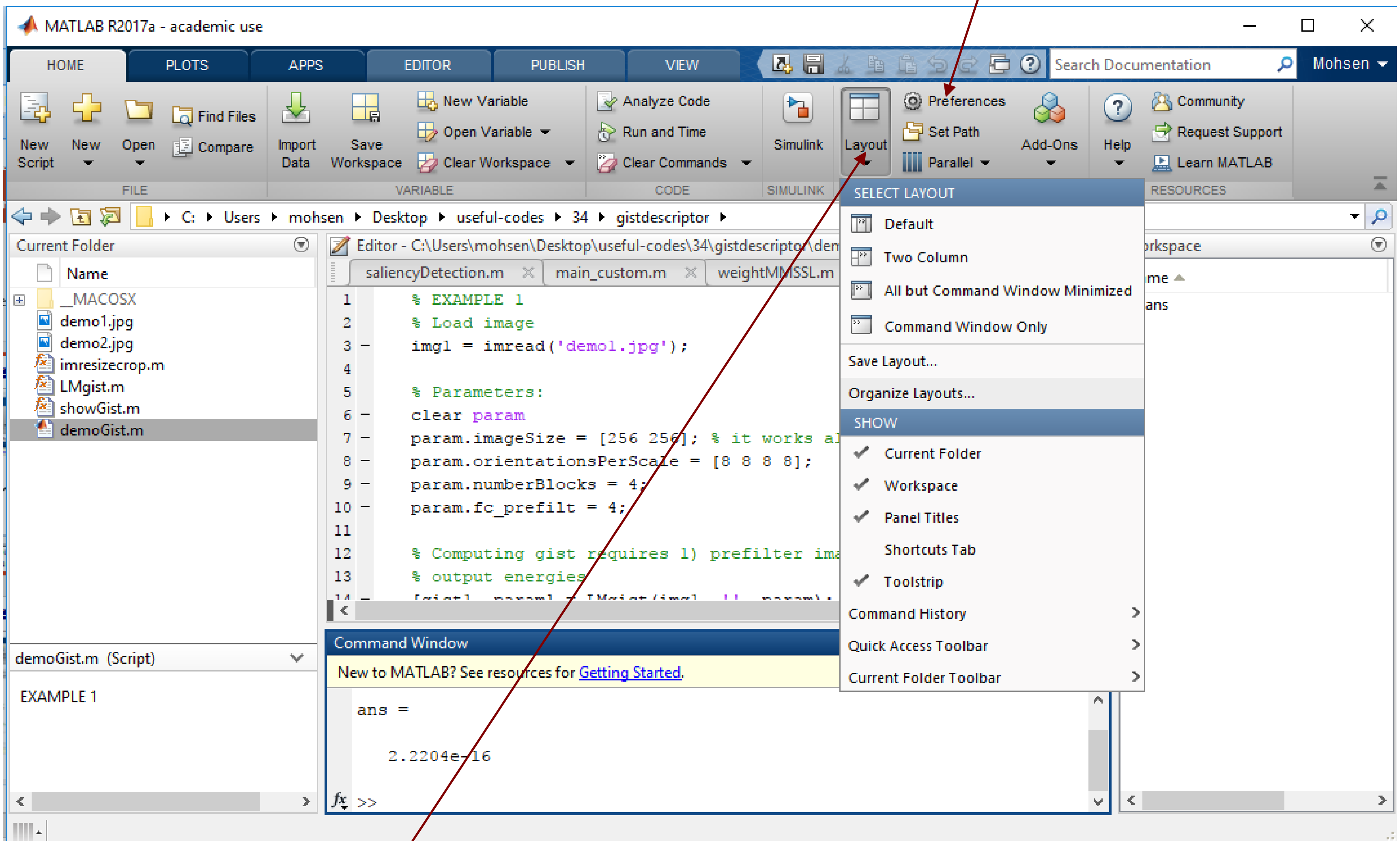
MATLAB Basics

1. MATLAB desktop
2. MATLAB editor
3. Getting help
4. Variables, built-in constants, keywords
5. Numbers and formats
6. Arrays and matrices
7. Operators and expressions
8. Functions – built-in and user-defined
9. Basic plotting
10. Function maxima and minima
11. Strings, cell arrays, **`fprintf`**

These should be enough to get you started. We will explore them further, as well as other topics, in the rest of the course.

1. MATLAB Desktop

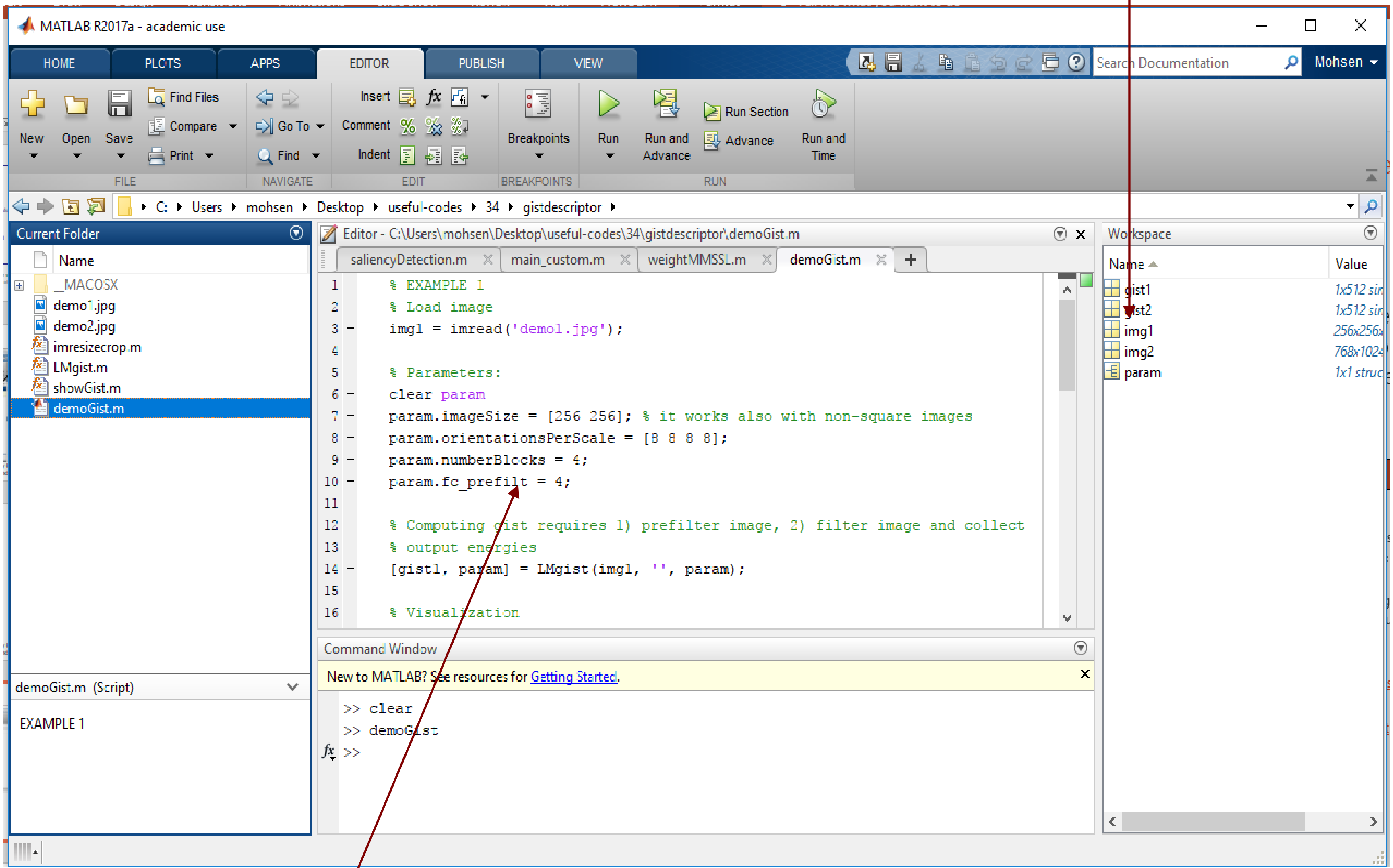
Setting



You can select what is on your desktop by Clicking on Layout. Go down to Command History and select docked.

2. MATLAB Desktop

workspace window



MATLAB Editor for writing Script Files or Functions

3. Getting Help

Several ways of getting help:

1) help menu item on MATLAB desktop opens up searchable help browser window

2) from the following commands:

comments begin with %

```
>> helpdesk              % open help browser
>> help topic            % e.g., help log10
>> doc topic             % e.g., doc plot
>> help                  % get list of all help topics
>> help dir              % get help on entire directory
>> help syntax           % get help on MATLAB syntax
>> help /                % operators & special characters
>> docsearch text        % search HTML browser for ‘text’
>> lookfor topic         % e.g., lookfor acos
```

4. Variables, Constants, Keywords

Variables require no special declarations of type or storage. Examples:

```
>> x = 3;                      % simple scalar
>> y = [4, 5, 6];              % row vector of length 3
>> z = [4; 5; 6];              % column vector of length 3
>> A = [1,2,3; 4,5,6];         % 2x3 matrix
>> s = 'abcd efg';                         % string
>> C = {'abc', 'defg', '123-456'};         % 1x3 cell array
```

math notation

$$y = [4, 5, 6]\,, \quad z = \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix}, \quad A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

the functions **class** and **size** tell you the type and dimensions of the defined object, e.g.,

```
>> class(C)
>> size(C)
```

```
>> x = 3
x =
     3

>> y = [4, 5, 6]
y =
     4     5     6

>> z = [4; 5; 6]          % note, z = y'
z =
     4
     5
     6

>> A = [1 2 3; 4 5 6]
A =
     1     2     3
     4     5     6
```

Several things happen with this simple MATLAB command:

A variable, x, of type double is created
A memory location for the variable x is assigned
The value 3 is stored in that memory location called x.

What are your variables? How to clear them?
Use workspace window, or the commands:

who, whos, clear, clc, close

```
>> who
Your variables are:
A  y  z

>> whos
  Name         Size         Bytes  Class     Attributes
  A            2x3             48  double
  y            1x3             24  double
  z            3x1             24  double

>> clear all      % clear all variables from memory
>> clc            % clear command window
>> close all      % close all open figures
```

Operating system commands:

```
>> path                 % display search path
>> pathtool             % modify search path
>> addpath dir          % add directory to path

>> cd dir               % change directory
>> pwd                  % print working directory

>> dir                  % list all files in current dir
>> what                 % list MATLAB files only
>> which file           % display location of file
>> edit file            % invoke MATLAB editor
>> help                 % command provides information
                            about a function
>> help sin             %This only works if you know
                        the name of the function you
                         want help with.
>> quit                 % quit MATLAB
>> exit                 % quit MATLAB
```

Some MATLAB® Math Functions

Function	MATLAB®		Function	MATLAB®
cosine	cos or cosd		square root	sqrt
sine	sin or sind		exponential	exp
tangent	tan or tand		logarithm (base 10)	log10
cotangent	cot or cotd		natural log (base e)	log
arc cosine	acos or acosd		round to nearest integer	round
arc sine	asin or asind		round down to integer	floor
arc tangent	atan or atand		round up to integer	ceil
arc cotangent	acot or acotd			

Note: cos(α) assumes α in radians; whereas, cosd(α) assumes α in degrees.
acos(x) returns the angle in radians; whereas, acosd(x) returns the angle in degrees.

π radians = 180 degrees

Naming Rules for Variables

1. Variable names must begin with a letter
2. Names can include any combinations of letters, numbers, and underscores
3. Maximum length for a variable name is 63 characters
4. MATLAB® is case sensitive. The variable name A is different than the variable name a.
5. Avoid the following names: i, j, pi, and all built-in MATLAB® function names such as length, char, size, plot, break, cos, log, …
6. It is good programming practice to name your variables to reflect their function in a program rather than using generic x, y, z variables.

Special built-in math constants that should not (though they can) be re-defined as variables:

```
eps         % machine epsilon - floating-point accuracy
i,j         % imaginary unit, i.e., sqrt(-1)
Inf,inf     % infinity
intmax      % largest value of specified integer type
intmin      % smallest value of specified integer type
NaN,nan     % not-a-number, e.g., 0/0, inf/inf
pi          % pi
realmax     % largest positive floating-point number
realmin     % smallest positive floating-point number
```

Note: **`i,j`** are commonly used for array and matrix indices. If you're dealing with complex-valued data, avoid redefining both **`i,j`**.

Values of special constants:

```
>> eps                  % equal to 2^(-52)
ans =
   2.2204e-016          % MATLAB’s floating-point accuracy
                        % i.e., 2.2204 * 10^(-16)
>> intmax               % 2^(31)-1 for 32-bit integers
ans =
   2147483647

>> intmin               % equal to -2^(31)
ans =
  -2147483648

>> realmax              % equal to (2-eps)*2^(1023)
ans =
   1.7977e+308          % i.e., 1.7977 * 10^(308)

>> realmin              % 2^(-1022) = 2.2251 * 10^(-308)
ans =
   2.2251e-308
```

Special keywords that cannot be used as variable names:

```
>> iskeyword

ans =
    'break'                 'function'
    'case'                  'global'
    'catch'                 'if'
    'classdef'              'otherwise'
    'continue'              'parfor'
    'else'                  'persistent'
    'elseif'                'return'
    'end'                   'switch'
    'for'                   'try'
                            'while'

'true' , 'false'
```

How Computers Store Variables

Computers store all data (numbers, letters, instructions, …) as strings of 1s and 0s (bits).

A **bit** is short for **bi**nary digi**t**. It has only two possible values: On (1) or Off (0).

A byte is simply a string of 8 bits.

A kilobyte (kB) is 1000 bytes (commercial), kilobyte is traditionally used to denote 1024 (2^{10}) bytes.

A megabyte (MB) is 1,000,000 bytes

A gigabyte (GB) is 1,000,000,000 bytes

For a sense of size, click on link below:

http://highscalability.com/blog/2012/9/11/how-big-is-a-petabyte-exabyte-zettabyte-or-a-yottabyte.html

5. Numbers and Formats

MATLAB by default uses double-precision (64-bit) floating-point numbers following the IEEE floating-point standard. You may find more information on this standard in:

Representation of Floating-Point Numbers

C. Moler, "Floating Points," MATLAB News and Notes, Fall, 1996 (PDF file)

x = (-1)^s * (1+f) * 2^(e-1023)

1 bit sign | 52 bits mantissa | 11 bits exponent

1 <= e <= 2046, e=0, e=2047

0 <= f < 1

f_min = eps = 2^(-52)

machine epsilon

MATLAB can also use single-precision (32-bit) floating point numbers if so desired.

There are also several integer data types that are useful in certain applications, such as image processing or programming DSP chips. The integer data types have 8, 16, 32, or 64 bits and are signed or unsigned:

```
 int8,   int16,   int32,   int64
uint8,  uint16,  uint32,  uint64
```

These data types work for integers as long as the integers don't exceed the range for the data type chosen.

They take up less memory space than doubles.

They don’t work for non-integers. If you create a variable that is an int8 and try to assign it a value of 14.8, that variable will be assigned a value of 15 instead (closest integer within the range).

One common application for integer data types is image data (jpeg, png, …)

For more information do:

```
>> help datatypes
>> help class          % determine datatype
>> help int32          % example
```

Numeric Data Types

MATLAB has several different options for storing numbers as bits. Unless you specify otherwise, all numbers in MATLAB are stored as doubles.

Name	Description	Range
double	64 bit floating point	$\pm 2.23\times10^{-308}$ to $\pm 1.80\times10^{308}$
single	32 bit floating point	$\pm 1.18\times10^{-38}$ to $\pm 3.4\times10^{38}$
uint8	8 bit unsigned integer	Integers from 0 to 255
int8	8 bit signed integer	Integers from –128 to 127
uint16	16 bit unsigned integer	Integers from 0 to 65535
int16	16 bit signed integer	Integers from –32768 to 32767
uint32	32 bit unsigned integer	Integers from 0 to 4294967295
int32	32 bit signed integer	Integers from –2147483648 to 2147483647

Why should I care how data is stored in a computer?

Example: Perform each of the following calculations in your head.

a = 4/3

b = a – 1

c = 3*b

e = 1 – c

What does MATLAB get?

Why should I care how data is stored in a computer?

What does MATLAB get?

a = 4/3 = 1.3333

b = a – 1 = 0.3333

c = 3*b = 1.0000

e = 1 – c = 2.2204e-016

It is not possible to perfectly represent all real numbers using a finite string of 1s and 0s.

Comments

Not all numbers can be represented exactly even using 64 bit doubles.

If we do many, many calculations with numbers which are just a tiny bit off, that error can grow very, very large depending on the type of computations being performed.

64 bit doubles have a huge, but still limited range.

What happens if we exceed it? Try the following:

```
>>  a = 373^1500
>>  b = factorial(172)
```

Complex Numbers

By default, MATLAB treats all numbers and expressions as complex (even if they are real).

No special declarations are needed to handle complex-number operations. Examples:

```
>> z = 3+4i;                 % or, 3+4j, 3+4*i, 3+4*j
>> x = real(z);              % real part of z
>> y = imag(z);              % inaginary part of z
>> R = abs(z);               % absolute value of z
>> theta = angle(z);         % phase angle of z in radians
>> w = conj(z);              % complex conjugate, w=3-4i
>> isreal(z);                % test if z is real or complex
```

$$z = x + j\,y = R\,e^{j\theta}\,, \quad R = |z| = \sqrt{x^2 + y^2}\,, \quad \theta = \arctan\frac{y}{x}$$

cartesian & polar forms

math notation: θ = Arg(z)

```
>> z = 3+4j
z =
   3.0000 + 4.0000i

>> x = real(z)
x =
     3
>> y = imag(z)
y =
     4
>> R = abs(z)
R =
     5
>> theta = angle(z)         % in radians
theta =
    0.9273

>> abs(z - R*exp(j*theta)) + abs(z-x-j*y)   % test
ans =
  6.2804e-016
```

equivalent definitions:

```
z = 3+4*j
z = 3+4i
z = 3+4*i
z = complex(3,4)
```

Display Formats

```
>> format                 % default - 4 decimal places
>> format short           % same as the default
>> format long            % 15 decimal places
>> format short e         % 4 decimal – exponential format
>> format short g         % 4 decimals – exponential or fixed
>> format long e          % 15 decimals - exponential
>> format long g          % exponential or fixed
>> format shorteng        % 4 decimals, engineering
>> format longeng         % 15 decimals, engineering
>> format hex             % hexadecimal
>> format rat             % rational approximation
>> format compact         % conserve vertical spacing
>> format loose           % default vertical spacing

>> vpa(x,digits)          % variable-precision-arithmetic
```

These affect only the display format – internally all computations are done with full (double) precision

Example - displayed value of **10*pi** in different formats:

```
31.4159                    % format, or format short
31.415926535897931         % format long
3.1416e+001                % format short e
31.416                     % format short g
3.141592653589793e+001     % format long e
31.4159265358979           % format long g
31.4159e+000               % format shorteng
31.4159265358979e+000      % format longeng

>> vpa(10*pi)              % symbolic toolbox
ans =
31.415926535897932384626433832795

>> vpa(10*pi,20)           % specify number of digits
ans =
31.415926535897932385
```

```
>> help format
>> help vpa
>> help digits
```

ASCII Code

When you press a key on your computer keyboard, the key that you press is translated to a binary code.

A = 1000001 (Decimal = 65)

a = 1100001 (Decimal = 97)

0 = 0110000 (Decimal = 48)

ASCII Code

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	□

Strings in MATLAB

MATLAB stores strings as an array of characters using the ASCII code.

Each letter in a string takes up two bytes (16 bits) and the two bytes are the binary representation of the decimal number listed in the ASCII table.

Try the following in MATLAB:

```
>> month = ‘August’
>> double(month)
```

input/output functions: **disp**, **input**

```
>> x = 10; disp('the value of x is:'); disp(x);
the value of x is:
    10

>> x = input('enter x: ')        % numerical input
enter x: 100                     % 100 entered by user
x =
   100
```

prompt string in single quotes

```
>> y = input('enter string: ', 's');  % string input
enter string: abcd efg
>> y = input('enter string: ')
enter string: 'abcd efg'
y =
abcd efg
```

string entered with no quotes
string entered in quotes

```
>> help fprintf
>> help sprintf
```

```
>> help disp
>> help input
>> help menu
```

6. Arrays and Matrices

arrays and matrices are the most important data objects in MATLAB

We discuss briefly:

a) row and column vectors
b) transposition operator, '
c) colon operator, **:**
d) equally-spaced elements, linspace
e) accessing array elements
f) dynamic allocation & de-allocation
g) pre-allocation

The key to efficient MATLAB programming can be summarized in three words:

vectorize, vectorize, vectorize

and avoid all loops

Compare the two alternative computations:

```
x = [2,-3,4,1,5,8];
y = zeros(size(x));
for n = 1:length(x)
   y(n) = x(n)^2;
end
```

```
x = [2,-3,4,1,5,8];
y = x.^2;
```

element-wise exponentiation **.^**

ordinary exponentiation **^**

```
answer: y = [4,9,16,1,25,64]
```

```
>> x = [0 1 2 3 4 5]              % row vector
x =
     0     1     2     3     4     5

>> x = 0:5                        % row vector
x =
     0     1     2     3     4     5

>> x = [0 1 2 3 4 5]'             % column vector, (0:5)'
x =
     0
     1
     2
     3
     4
     5
```

the prime operator, ', or transpose, turns row vectors into column vectors, and vice versa

caveat: ' is actually conjugate transpose, use dot-prime, .' , for transpose w/o conjugation

```
>> z = [i; 1+2i; 1-i]      % column vector
z =
        0 + 1.0000i
   1.0000 + 2.0000i
   1.0000 - 1.0000i

>> z.'                     % transpose without conjugation
ans =
        0 + 1.0000i   1.0000 + 2.0000i   1.0000 - 1.0000i

>> z'                      % transpose with conjugation
ans =
        0 - 1.0000i   1.0000 - 2.0000i   1.0000 + 1.0000i

>> (z.')'                  % same as (z').' , or, conj(z)
ans =
        0 - 1.0000i
   1.0000 - 2.0000i
   1.0000 + 1.0000i
```

about linspace:

```
x = linspace(a,b,N+1);
```

is equivalent to:

```
x = a : (b-a)/N : b;
```

i.e., N+1 equally-spaced points in the interval [a,b]
or, dividing [a,b] into N equal sub-intervals

$$x(n) = a + \left(\frac{b-a}{N}\right)(n-1)\,, \quad n = 1,2,\ldots,N+1$$

step increment

```
>> x = 0 : 0.2 : 1              % in general, x = a:s:b
>> x = linspace(0,1,6)          % see also logspace
x =
     0   0.2000   0.4000   0.6000   0.8000   1.0000
```

6 points, 5 subintervals

```
>> x = 0 : 0.3 : 1
x =
     0    0.3    0.6    0.9


>> x = 0 : 0.4 : 1
x =
     0    0.4    0.8


>> x = 0 : 0.7 : 1
x =
     0    0.7
```

step increment

```
x = a : s : b;
```

the number of subintervals within [a,b] is obtained by rounding **`(b-a)/s`**, down to the nearest integer,

```
N = floor((b-a)/s);
```

`length(x)` is equal to **`N+1`**

```
x(n) = a + s*(n-1),
n = 1,2,...,N+1
```

```
% before rounding, (b-a)/s was in the three cases:
% 1/0.3 = 3.3333,  1/0.4 = 2.5,  1/0.7 = 1.4286
```

Note: MATLAB array indices always start with 1 and may not be 0 or negative

exception: logical indexing, discussed later

```
>> x = [ 2,    5,   -6,   10,    3,    4 ];
```

```
x(1), x(2), x(3), x(4), x(5), x(6)
```

Other languages, such as C/C++ and Fortran, allow indices to start at 0. For example, the same array would be declared/defined in C as follows:

```
double x[6] = { 2,    5,   -6,   10,    3,    4 };
```

```
x[0], x[1], x[2], x[3], x[4], x[5]
```

accessing array entries:

```
>> x = [2, 5, -6, 10, 3, 4]
x =
     2     5    -6    10     3     4

>> length(x)       % length of x, see also size(x)
ans =
     6

>> x(1)            % first entry
ans =
     2

>> x(3)            % third entry
ans =
    -6

>> x(end)          % last entry – need not know length
ans =
     4
```

accessing array entries:

```
>> x(end-3:end)            % x = [2, 5, -6, 10, 3, 4]
ans =
     -6   10      3      4       % last four

>> x(3:5)                  % list third-to-fifth entries
ans =
    -6    10      3

>> x(1:3:end)              % every third entry
ans =
     2    10

>> x(1:2:end)              % every second entry
ans =
     2    -6      3
```

accessing array entries:

```
>> x = [2, 5, -6, 10, 3, 4];

>> x(end:-1:1)    % list backwards, same as fliplr(x)
ans =
     4   3   10   -6    5    2


>> x([3,1,5])        % list [x(3),x(1),x(5)]
ans =
     -6      2      3

>> x(end+3) = 8
x =
    2     5    -6    10     3     4     0     0     8
```

automatic memory re-allocation

automatic memory allocation and de-allocation:

```
>> clear x

>> x(3) = -6
x =
     0     0    -6

>> x(6) = 4
x =
     0     0    -6     0     0     4

>> x(end) = []                    % delete last entry
x =
     0     0    -6     0     0

>> x = [2, 5, -6, 10, 3, 4];
>> x(3)=[]                        % delete third entry
x =
     2     5     10     3     4
```

pre-allocation

```
>> clear x
>> x = zeros(1,6)                   % 1x6 array of zeros
x =
     0     0     0     0     0     0

>> x = zeros(6,1)                   % 6x1 array of zeros
x =
     0
     0
     0
     0
     0
     0
```

```
>> help zeros
>> help ones
```

illustrating dynamic allocation & pre-allocation

```
clear x;
for k=[3,7,10]                 % k runs successively through
   x(k) = 3 + 0.1*k;           % the values of [3,7,10]
   disp(x);                    % diplay current vector x
end
```

```
    0.0  0.0  3.3
    0.0  0.0  3.3  0.0  0.0  0.0  3.7
    0.0  0.0  3.3  0.0  0.0  0.0  3.7  0.0  0.0  4.0
```

```
x = zeros(1,10);               % pre-allocate x to length 10
for k=[3,7,10]
   x(k) = 3 + 0.1*k;
   disp(x);
end
```

```
    0.0  0.0  3.3  0.0  0.0  0.0  0.0  0.0  0.0  0.0
    0.0  0.0  3.3  0.0  0.0  0.0  3.7  0.0  0.0  0.0
    0.0  0.0  3.3  0.0  0.0  0.0  3.7  0.0  0.0  4.0
```

First assignment will be posted on the course website on Saturday 28.10.2017

Due date: Wed**nes**day 1.11.2017 11:55 PM

Try to use LaTeX for generating your report and send me the PDF

Recommended: Use the Overleaf online website for generating latex documents: https://www.overleaf.com